Programming with FORTRAN and/or C

I. Objective

To learn the basic commands needed to write FORTRAN and/or C programs.

II. Introduction

Up to this point we have seen that many interesting physics problems may be solved within available software packages without any programming at all. However, there are several occasions when obtaining a numerical solution requires one to write a program:

- When the problem is very specialized. Research often presents problems that are very specific in nature for which software packages do not exist.
- When the problem is too big for existing software. The solution of a 100 x 100 set of coupled linear equations is typically too big for Maple. In this case writing a program would make sense.
- When the problem requires high computation speed. A program dedicated to a single task will typically run faster than a general application designed to handle many different problems.

In today's exercises we will extend what you learned in Physics 221 for wave motion in a wire of uniform mass density to the case of a non-uniform wire. We will solve the problem numerically by writing a program in FORTRAN or C on the Project Vincent workstations, and calculate the frequencies and wave functions of the resonant standing wave modes. We begin by showing some essential operations in the UNIX environment.

III. Exercises

A. Running FORTRAN or C Programs in the UNIX Environment

Objective:	To gain familiarity with compiling and running FORTRAN or C programs, and graphing data on Vincent.
Where to begin:	on Project Vincent
What to do:	follow the instructions below
What to turn in to your instructor:	nothing: for your information only
What to put in log book:	any problems you have and how you solve them

(1) Copy Files from the Physics 232 Locker: We placed several files you may use for today's activities in the Physics 232 locker. Go to the locker (to access the locker type **add physics** and then go to the directory **/home/physics/phys232**). Copy the files **in (**or **in.txt), aaa, numerov.f,** and **test.f** (or **numerov.c** and **test.c**) to your working directory.

(2) Running FORTRAN Programs: Suppose we wish to run the FORTRAN program **test.f** (note that FORTRAN program names must end with the extension **.f**) that reads a variable from the input file **aaa**. First the program must be **compiled** and **linked**. This is done by typing:

f77 -o testf test.f

Command	What it does
f77 -o testf test.f	compiles the FORTRAN program **test.f,** links the object file, and creates an executable file called **testf**.

Running C Programs: Suppose we wish to run the C program **test.c** (note that C program names must end with the extension **.c**) that reads a variable from the input file **aaa**. First the program must be **compiled** and **linked**. This is done by typing:
cc -o testc test.c

Command	What it does
cc -o testc test.c	compiles the C program **test.c**, links the object file, and creates an executable file called **testc**.

The program **testf** (or **testc**) calculates the quantity aN^2, where *a* is a number specified in the input file **aaa** and *N* is an integer from 0 to 10. Set *a* to the value 1 by editing **aaa** using the command: **joe aaa**

Command	What it does
joe aaa	uses the **joe** editor to open the file **aaa**. The editor commands may be displayed by typing **^k^h (control-k control-h).**

To run the FORTRAN program and store the data in the file **fort.8**, type: **testf < aaa**
To run the C program and store the data in the file **out**, type: **testc < aaa > out**

Command	What it does
testf < aaa	runs the executable file **testf** which reads the input file **aaa**.
testc < aaa > out	runs the executable file **testc** which reads **aaa** and writes to **out**.

(3) Graphing Data Files: The output of the program was written to a file called **fort.8** (or **out**). To graph the contents of this two column file type: **xgraph fort.8** (or **xgraph out**).

Command	What it does
xgraph fort.8	graphs the data in **fort.8**.
xgraph out	graphs the data in **out.**

B. Waves in a Wire

Objective:	to outline the physics of today's problem.
What to do:	read the material below and derive the stated equations.
What to turn in to your instructor:	your log book.
What to put in log book:	derivations from this exercise.

(1) Finding vibrational frequencies in a uniform wire using the shooting method: In PHY 221 (Serway, 22.5) you learned that a wave traveling in a uniform wire of linear mass density under a constant tension *T* has a frequency given by $f = \frac{k}{2}\sqrt{\frac{T}{}}$, where the wave number *k* is defined as $k = \frac{2}{}$. The transverse displacement *y* of the wire as a function of position *x* is given by $\frac{d^2y}{dx^2} + k^2y = 0$.

If the wire is fixed at both ends then one has the **boundary conditions** $y(0)=y(L)=0$ and one finds that resonant standing waves in the wire occur for discrete values of k (since L needs to be a multiple of $\lambda/2$), and therefore discrete frequencies $f = \frac{k}{2}\sqrt{\frac{T}{\mu}}$. If one wanted to numerically determine those values of k for which the resonant standing waves occur, one could start with the two boundary conditions at the left end of the wire, $y(0)$ and $y'(0)$, **guess an approximate value** *of* k, numerically solve for the function y over the interval $[0,L]$, and then check to see how close $y(L)$ was to 0, the desired boundary condition. If the difference were not acceptable, a new guess for k could be made and a new $y(L)$ calculated until $y(L) \approx 0$ within an acceptable tolerance. This trial and error approach is called the **shooting method** and is the method we will use to determine the resonant frequencies of a non-uniform wire (which we cannot calculate analytically).

(2) Vibrations of a non-uniform wire: Suppose that two pieces of wire of equal length with masses per unit length μ_1 and μ_2 are joined to form one piece of total length L=1 m. The wire is then placed under a tension T and its ends are fixed, as shown in the figure. Suppose that μ_1=0.001 kg/m, μ_2=0.002 kg/m, L=1 m, and T=100 N . Suppose also that our coordinates are as shown

x_{init}=0.0 m x_{mid}=0.5 m x_{fin}=1.0 m

in the Figure above. We wish to study the resonant standing waves in this wire and determine the frequencies of the vibrational modes.

(3) Defining the problem mathematically: We therefore wish to solve the differential equation

$$\frac{d^2y}{dx^2} + k(x)^2 y = 0$$

for the case of the non-uniform wire, where $k(x)$ is a function that may be written as

$$k(x) = \begin{cases} k_1 & \text{if } 0 \le x \le \frac{1}{2} \\ k_1\sqrt{\frac{\mu_2}{\mu_1}} & \text{if } \frac{1}{2} < x \le 1 \end{cases}$$

Since the differential equation does not have constant coefficients (i.e., k is a function of x), we can only solve it numerically. We will use the boundary values $y(0)=0$ and $y'(0)$, along with an initial guess for k, to solve for the values of y over the interval [0,1].

(4) Derivation: Show in your log book that, because the two parts of the wire vibrate with the same frequency, the following formula holds:

$$\frac{k_2}{k_1} = \sqrt{\frac{\mu_2}{\mu_1}}$$

C. Numerical Solution Using Numerov Algorithm

Objective:	to set up the algorithm we will use to solve the differential equation $\frac{d^2y}{dx^2}+k^2y=0$.
Where to begin:	here.
What to do:	follow the instructions below.
What to turn in to your instructor:	your log book.
What to put in log book:	derivation of the equations specified below.

(1) The Algorithm: Once the differential equation and the boundary conditions at some point are determined, the differential equation may be solved numerically. Differential equations of the general form

$$\frac{d^2y}{dx^2}+k^2(x)y=S(x)$$

may be solved using the **Numerov algorithm** (see Appendix 1 for the recursion formulas). In your log book, show that the Numerov recursion formulas for the case of *S(x)=0* become

$$y_1=y_0\left(1-\frac{h^2k_0^2}{2}\right)+y_0'h$$

$$y_i=\frac{1}{\left(1+\frac{h^2k_i^2}{12}\right)}\left\{2\left(1-\frac{5h^2k_{i-1}^2}{12}\right)y_{i-1}-\left(1+\frac{h^2k_{i-2}^2}{12}\right)y_{i-2}\right\}$$

D. Writing the Program

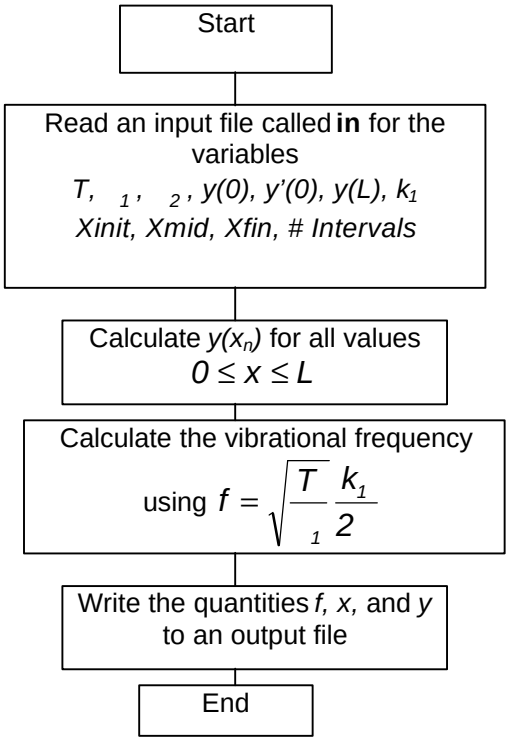
Objective:	to write a FORTRAN (or C) program to numerically determine the frequencies and wave functions for the resonant standing waves in a non-uniform wire.
Where to begin:	on Vincent.
What to do:	follow the instructions below.
What to turn in to your instructor:	your log book.
What to put in log book:	the time you begin your work, problems you encounter, solutions you find.

Keep in mind the following general rules for programming in FORTRAN and C:

FORTRAN Rules
A letter **C** in the first column of a line indicates that a comment will follow. The FORTRAN compiler will skip the rest of the line (ignore it).
Array dimensions must be specified at the beginning of a FORTRAN routine.
Variables may be declared at the beginning of a FORTRAN routine. Undeclared variable names that begin with letters A-H and O-Z are understood to be real and single precision. Undeclared variables beginning with letters I-N are understood to be integers.
Only one statement per line, fixed format of program.
Indent all FORTRAN statements at least six spaces (except when using labels to identify lines).

C Rules
Comments begin with **/*** and end with the next ***/**. They are skipped (ignored) by the compiler.
Each program must have exactly one function called **int main(int argc, char *argv[]).**
All variables (simple and arrays) and functions must be declared at the beginning of a block.
All statements must end with a semicolon **;** More than statement per line allowed.
Blocks (including functions) must begin and end with curly brackets **{ }**.

(1) Planning the Program: The program you write should have the following general outline:



(2) Creating the Main Program: To begin writing the main program, type the command **joe num.f** (or **joe num.c**). Type the following lines (for FORTRAN):

```
ccccccccccccccccccccccccccccccccccccccccccccccccccc
c       Numerov Algorithm
ccccccccccccccccccccccccccccccccccccccccccccccccccc
              dimension x(10000),y(10000)
              character a
              common/one/ xk1,xk2,xinit,xmid,xfin
```

Statement	What it does
dimension x(10000),y(10000)	creates two arrays **x** and **y** of dimension 1x10000 (i.e., vectors with 10000 components).
character a	creates a variable **a** that contains ASCII characters (letters and symbols).
common /one/	makes the variables following the **common** statement available to all subroutines in the program that include the same **common** statement. The symbol **/one/** is a name (any other name will do as long as it begins with a letter) that distinguishes different **common** statements (though in this program there is only one).

Type the following lines as your skeleton for the Numerov program in C:

```
#include <stdio.h>
#include <math.h>
double xk1=0.0, xk2=0.0, xinit=0.0, xmid=0.0, xfin=0.0;
double wav(double);
void Numerov(double, double, double, double, int, double*,
double[], double[]);
/*************************************************************
***********/
int main(int argc, char *argv[])
{
   double x[10000], y[10000];
   double T=0.0, rho1=0.0, rho2=0.0, y0=0.0, Dy0=0.0, yL=0.0;
   int N=0, i=0;
   double pi=3.1415926, ymax=0.0, delta=0.0, freq=0.0;
   char line[100];

   return 0;
}
```

Statement	What it does
#include <...>	Include headers for math functions (**sqrt**) or file input/output.
double, int, void	define variables, either global or in a block **{ }**, may have initializers **=**
double wav(double);	Function prototype, the actual code comes later in the file.
{ }	Curly brackets mark the begiining and end of a block (e.g., function)
int main(int, char*)	Main function. Program execution starts here.
return 0;	Return to calling function (or the operating system).
char line[100];	Allocate an array of characters with 100 elements.

(3) Reading the Input File: The input file **in** contains all of the information about the physical parameters, boundary conditions, and the number line. If you view the contents of the file (type **more in**), you will see several lines of text separating lines of numbers. The text in the file serves only to remind us what the numbers represent. In reading the file, therefore, we wish only to extract the numbers. To do this type the following lines (for FORTRAN):

```
          read(5,10) a,a,a
           read(5,*) T,rho1,rho2
           read(5,10) a,a,a,a
           read(5,*) y0,Dy0,yL
           read(5,10) a,a,a,a
           read(5,*) xk1
           read(5,10) a,a,a,a
           read(5,*) xinit,xmid,xfin,N
10        format(a1)
```

Statement	What it does
read(5,10) a,a,a **10 format(a1)**	opens the input file (which is designated by the **5**), and reads three character strings that are one character long (specified by the symbol **a1**). This is how the text lines are skipped.
read(5,*) T,rho1,rho2	reads the numbers and assigns them to the variables T, rho1, and rho2

For the C program, type the following lines:

```
FILE *i_file;                          /* declare variable for file I/O */
i_file=fopen("in.txt","r");            /* open file for reading */
for (i=0; i<3; i++) fgets(line,sizeof(line),i_file); /* skip 3 lines */
fgets(line,sizeof(line),i_file);   /* read one line, store in line */
sscanf(line,"%lf %lf %lf",&T, &rho1, &rho2);
                                       /* parse line and read variables */
for (i=0; i<4; i++) fgets(line,sizeof(line),i_file);
fgets(line,sizeof(line),i_file);
sscanf(line,"%lf %lf %lf", &y0, &Dy0, &yL);
for (i=0; i<4; i++) fgets(line,sizeof(line),i_file);
fgets(line,sizeof(line),i_file);
sscanf(line,"%lf",&xk1);
for (i=0; i<4; i++) fgets(line,sizeof(line),i_file);
fgets(line,sizeof(line),i_file);
sscanf(line,"%lf %lf %lf %d",&xinit,&xmid,&xfin,&N);
fclose(i_file);                        /* close file, we are done */
```

Statement	What it does
FILE *i_file;	Define a variable (pointer to file) for file I/O.
i_file=fopen("in.txt","r");	Open file with given name for reading only (**"r"**)
for (i=0; i<3; i++) block	Repeat block for **i=0**, **i=1**, and **i=2**. Start with **i=0;** continue as long as **i<3**; increment i by 1 (**i++**) after each loop. **block** is either one statement or many statements enclosed by **{ }** .
fgets(line,n,i_file);	read one line from **i_file**, store in **line**, no more than **n** characters.
sizeof(line)	the maximum number of characters to be stored in line (100)
sscanf(line,format,pointers**)**	Parse **line** according to format and store variables in pointers
%lf	format string for a long floating point number (double precision)
&T	pointer to **T**, i.e., the address of **T**, needed to return parameters
fclose(i_file);	close the file.

(4) Performing the Calculations: We now wish to write the part of the program that calculates the function *y(x),* the frequency *f*, and writes the values of *x* and *y* to an output file. Type the following (for the FORTRAN program):

```
xk2=xk1*sqrt(rho2/rho1)
pi=3.1415926
ymax=0.
call numerov(xinit,y0,Dy0,xfin,N,ymax,x,y)

delta=(y(N)-yL)/ymax
freq=sqrt(T/rho1)*xk1/2/pi

write(7,*) 'Delta = ',delta
write(7,*) 'Frequency = ',freq
do i=0,N
        write(7,*) x(i),y(i)
enddo
end
```

Statement	What it does
call numerov( )	Calls a subroutine **numerov** that performs the calculations of the Numerov algorithm. The variables **xinit, y0, Dy0, xfin,** and **N** are known at the time of the call and are passed to the routine. The variables **ymax, x, y** are calculated by the subroutine and passed back to the main program. Note **x** and **y** are arrays.
delta=(y(N)-yL)/ymax	calculates by how much the final value of **y** misses the specified boundary condition **yL** as a fraction of the maximum value of the wave function
freq=sqrt(T/rho1)*xk1/2/pi	calculates the vibration frequency from the relation $f = \sqrt{\frac{T}{1}}\frac{k_1}{2}$
write(7,*) 'Delta=',delta	writes the value of **delta** to an output file that FORTRAN will call **fort.7**
do i=0,N **write(7,*) x(i),y(i)** **enddo**	a **do loop** that writes the values of **x** and **y** to the file **fort.7** for values of the index **i** from 0 to N (the indentation in front of the **write** is not necessary, but makes the program easier to read).
end	signals that the main program has ended; defines the boundary of the program to which the variable names apply

In C, you might write the following:

```
xk2=xk1*sqrt(rho2/rho1);
Numerov(xinit,y0,Dy0,xfin,N,&ymax,x,y);
delta=(y[N]-yL)/ymax;
freq=sqrt(T/rho1)*xk1/2/pi;
printf("Delta     = %lf\n",delta);
printf("Frequency = %lf\n",freq);
for (i=0; i<=N; i++) {   printf("%lf %lf\n",x[i],y[i]);   }
```

Statement	What it does
Numerov(xinit,y0,Dy0,xfin,N ,&ymax,x,y);	calls a function **Numerov()** that performs the calculations of the Numerov algorithm. The variables **xinit, y0,Dy0,xfin,** and **N** are known at the time of the call, and are passed by value. The variables **ymax, x, y** are calculated by the functions and passed back to the main program through pointers (by reference). Note that **x** and **y** are arrays, therefore they are passed using pointers by default.
delta=(y[N]-yL)/ymax;	calculates by how much the final value of **y** misses the specified boundary condition **yL** as a fraction of the maximum value of the wave function
freq=sqrt(T/rho1)*xk1/2/pi;	calculates the vibration frequency from the relation $f = \sqrt{\frac{T}{1}}\frac{k_1}{2}$
printf("Delta = %lf\n", delta);	writes the value of **delta** to the standard ouput file (screen). Use piping **>** to write to a file on your disk. **\n** means new line.
for (i=0; i<=N; i++) { printf("%lf %lf\n",x[i],y[i]); }	a **for** loop that writes the values of **x** and **y** to the file **std** for values of the index **i** from **0** to **N**. Note the format string **%lf**.

	Array elements are addressed using angular brackets **[ ]**.

(5) Writing the Numerov Subroutine: We now wish to write the subroutine that performs the recursion relation of the Numerov algorithm. The subroutine is written to accept the initial vales **xi, yi, Dyi, xf, N** and to calculate **x** and **y** and determine the maximum y value **ymax**. One way to write the subroutine (in FORTRAN) is as follows:

```
      subroutine numerov(xi,yi,Dyi,xf,N,ymax,x,y)
      dimension x(10000),y(10000)
      dx=(xf-xi)/N

      do i=0,N
            if(i.eq.0) then
                  x(0)=xi
                  y(0)=yi
                  if (abs(y(0)).gt.ymax) ymax=abs(y(0))
            else if(i.eq.1) then
                  x(1)=x(0)+dx
                  xk=wav(x(0))
                  y(1)=yi*(1-dx*dx*xk*xk/2)+Dyi*dx
                  if (abs(y(1)).gt.ymax) ymax=abs(y(1))
             else
                  x(i)=x(i-1)+dx
                  xk0=wav(x(i))
                  xk1=wav(x(i)-dx)
                  xk2=wav(x(i)-2*dx)
                  y(i)=1/(1+dx*dx*xk0*xk0/12)*
     +                  (2*(1-5*dx*dx*xk1*xk1/12)*y(i-1)
     +                  -(1+dx*dx*xk2*xk2/12)*y(i-2))
                  if (abs(y(i)).gt.ymax) ymax=abs(y(i))
            endif
      enddo
      end
```

Statement	What it does
subroutine numerov ( )	begins the subroutine **numerov**
dimension x(10000),y(10000)	dimension statements are needed in all subroutines that use arrays

if(i.eq.0) then
(process A)
else if(i.eq.1) then
(process B)
else
(process C)
endif
	this is a **conditional**. If i=0 then process A is performed and the program goes to **endif.** If i=1 then process B is performed and the program goes to **endif**. Otherwise process C is performed and the program goes to **endif**

xk=wav(x(i))	determines the value of the wave number **wav** at the point **x(i)** and assigns the number to the variable **xk**
y(i)=1/(1+dx*dx*xk0*xk0/12)* **+ (2*(1-5*dx*dx*xk1*xk1/12)*y(i-1)** **+ -(1+dx*dx*xk2*xk2/12)*y(i-2))**	calculates the Numerov recursion relation. Note that when a formula will not fit on one line, it may be wrapped over several lines by placing any desired character (in this case **+**) in the sixth column
dx*dx	I have used **dx*dx** instead of **dx^2**. The first may be calculated more quickly than the second.

return or **end**	all subroutines in FORTRAN must end with these lines

In C, the Numerov subroutine could be implemented as follows:

```
void Numerov(double xi, double yi, double Dyi, double xf, int N,
             double *ymax, double x[], double y[])
{
   double dx=0.0, xk=0.0, xxk0=0.0, xxk1=0.0, xxk2=0.0;
   int i=0;
   dx=(xf-xi)/N;

   for (i=0; i<=N; i++) {
      if (i==0) {
         x[0]=xi;
         y[0]=yi;
         if (fabs(y[0])>*ymax) *ymax=fabs(y[0]);
      } else if (i==1) {
         x[1]=x[0]+dx;
         xk=wav(x[0]);
         y[1]=yi*(1-dx*dx*xk*xk/2)+Dyi*dx;
         if (fabs(y[1])>*ymax) *ymax=fabs(y[1]);
      } else {
         x[i]=x[i-1]+dx;
         xxk0=wav(x[i]);
         xxk1=wav(x[i]-dx);
         xxk2=wav(x[i]-2*dx);
         y[i]=1/(1+dx*dx*xxk0*xxk0/12)*
                (2*(1-5*dx*dx*xxk1*xxk1/12)*y[i-1]
                -(1+dx*dx*xxk2*xxk2/12)*y[i-2]);
         if (fabs(y[i])>*ymax) *ymax=fabs(y[i]);
      }
   }
   return;
}
```

Statement	What it does
void Numerov ( )	Defines function N**umerov** of type void (no function result returned) and parameters passed.
double x[], double y[]	The arrays can have any length
if (i==0) { (block A) **} else if (i==1) {** (block B) **} else {** (block C) **} /* endif */**	This is a **conditional statement**. If i=0 then block A is executed. If i=1, then block B is executed. Otherwise block C is executed. Note that an equal-to condition is written as **(i==1)** to distinguish between condition and assignment **i=1**. The condition is enclosed in **( )**. The statements to be executed (blocks) are enclosed by **{ }**.
xk=wav(x[0]);	determines the value of the wave number **wav** at the point **x[i]** and assigns the number to the variable **xk**
y[i]=1/(1+dx*dx*xxk0*xxk0/12)* (2*(1-5*dx*dx*xxk1*xxk1/12)*y[i-1] -(1+dx*dx*xxk2*xxk2/12)*y[i-2]);	Calculates the Numerov recursion relation. Note that when a formula will not fit on one line, it may be wrapped over several lines without problems. Note that we have added variables **xxk0**, **xxk1**, **xxk2** not in the FORTRAN code.
dx*dx	C does not have an exponent operator **^** like FORTRAN.
return;	Return to calling function.

(6) Writing the function wav(): The last task is to define the function *k(x).* This will be calculated in the function **wav().** One way to do this (in FORTRAN) is shown below:

```
ccccccccccccccccccccccccccccccccccccccccccccccccccc
          function wav(x)
          common/one/ a,b,c,d,e

          if (x.ge.c.and.x.le.d) then
                 wav=a
          else if (x.ge.d.and.x.le.e) then
                 wav=b
          endif
          return
          end
```

Statement	What it does
function wav (x)	begins the function subroutine **wav**
common/one/ a,b,c,d,e	makes the variables **xk1, xk2, xinit, xmid, xfin** in the main routine common (accessible) to the function subroutine. The assignment is in the order listed so that **a=xk1, b=xk2, c=xinit, d=xmid, e=xfin**
return or **end**	all subroutines in FORTRAN must end with these lines

In C, you could simply write

```
double wav(double x)
{
   if ((x>=xinit) && (x<=xmid)) return xk1;
   else if ((x>=xmid) && (x<=xfin)) return xk2;
   else return 0.0;
}
```

Statement	What it does
double wav (double x)	In C, functions and subroutines are the same.
double xk1=0.0, xk2=0.0, xinit=0.0, xmid=0.0, xfin=0.0;	The variables **xk1, xk2, xinit, xmid, xfin** were defined as global variables at the very beginning, therefore they are known to all functions in the program. No COMMON blocks!
else return 0.0;	Since the block to be executed here has only one statement, we do not need the { } brackets (but they are allowed).
&&	logical AND operator

E. Running the Program

Objective:	To determine the frequencies and wave functions for the resonant standing waves in a non-uniform wire.
Where to begin:	On Vincent.
What to do:	Follow the instructions below.
What to turn in to your instructor:	Your log book, including the standing wave frequencies you determine, graphs of the modes of vibration.
What to put in log book:	The time you begin your work, problems you encounter, solutions you find.

(1) Compiling and running the program: In FORTRAN, you simply type

f77 -o numerov numerov.f

to compile and link the program.

In C, you need to say

cc -o numerov numerov.c /usr/lib/libm.a

otherwise the linker (loader) will not find the math library **libm.a** that contains the square root. (Don't you hate UNIX by now! Borland or IBM would never expect you to type this on your PC).

(2) Checking the Results: There are several ways the program may be checked. First, if the wire were uniform, you learned in Physics 221 that the standing wave will have *k* values given by $k_n = n$ and $y(x)=\sin(k_n x)$. Make the necessary changes in the and *k* values of the input file and check if you get the expected results.

Think of another check you can do to test your program. Comment on your results.

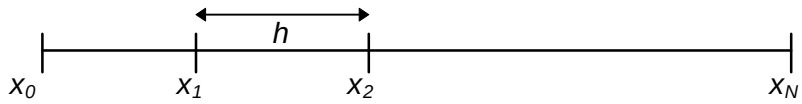
(3) Finding the Resonant Frequencies for the Non-uniform Wire: For *N=100*, run your program for the parameter values given in problem III.B.1. Use the shooting method described in III.B.2 to determine the first three resonant frequencies of the system. You may use **xgraph** to plot your results and see how close your guess for k_1 brought you to the final boundary condition. Print out the graphs for the first three standing wave modes.

IV. Appendix 1: The Numerov Algorithm

Equations of the form

$$\frac{d^2y}{dx^2} + k^2(x)y = S(x)$$

may be solved for $y(x)$ for points on the axis using the **Numerov Algorithm**[1] if the boundary conditions $y(x_0)$, $y'(x_0)$ are known at some point x_0.. The axis is divided into *N* intervals of equal length $h = \frac{x_N - x_0}{N}$ as shown below. This method may be summarized by the recursion



relations in the table below. We use the notation $y_i \equiv y(x_i)$, $y_i' = y'(x_i)$ and similar notation for k_i and S_i.

Numerov Algorithm Recursion Relations for y_0, y_1, and y_i
$$y_0 = y(x_0)$$
$$y_1 = y_0 + y_0' h + \frac{(-k_0^2 y_0 + S_0)}{2} h^2$$
$$y_i = \frac{1}{\left(1 + \frac{h^2 k_i^2}{12}\right)} \left\{ \frac{h^2}{12}(S_i + 10S_{i-1} + S_{i-2}) + 2\left(1 - \frac{5h^2 k_{i-1}^2}{12}\right) y_{i-1} - \left(1 + \frac{h^2 k_{i-2}^2}{12}\right) y_{i-2} \right\}$$

[1] See S. E. Koonin, *Computational Physics,* Chapter 3, for a complete description.

V. Appendix 2: Complete FORTRAN Code for Numerov Program

```
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c       Numerov Algorithm
c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
             dimension x(10000),y(10000)
             character a
             common/one/ xk1,xk2,xinit,xmid,xfin

             read(5,10) a,a,a
             read(5,*) T,rho1,rho2
             read(5,10) a,a,a,a
             read(5,*) y0,Dy0,yL
             read(5,10) a,a,a,a
             read(5,*) xk1
             read(5,10) a,a,a,a
             read(5,*) xinit,xmid,xfin,N
10         format(a1)

             xk2=xk1*sqrt(rho2/rho1)
             pi=3.1415926
             ymax=0.
             call numerov(xinit,y0,Dy0,xfin,N,ymax,x,y)

             delta=(y(N)-yL)/ymax
             freq=sqrt(T/rho1)*xk1/2/pi
             write(7,*) 'Delta = ',delta
             write(7,*) 'Frequency = ',freq
             do i=0,N
                     write(7,*) x(i),y(i)
             enddo
             end
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
             subroutine numerov(xi,yi,Dyi,xf,N,ymax,x,y)
             dimension x(10000),y(10000)

             dx=(xf-xi)/N
             do i=0,N
                     if(i.eq.0) then
                             x(0)=xi
                             y(0)=yi
                             if (abs(y(0)).gt.ymax) ymax=abs(y(0))
                     else if(i.eq.1) then
                             x(1)=x(0)+dx
                             xk=wav(x(0))
                             y(1)=yi*(1-dx*dx*xk*xk/2)+Dyi*dx
                             if (abs(y(1)).gt.ymax) ymax=abs(y(1))
                      else
                             x(i)=x(i-1)+dx
                             xk0=wav(x(i))
                             xk1=wav(x(i)-dx)
                             xk2=wav(x(i)-2*dx)
                             y(i)=1/(1+dx*dx*xk0*xk0/12)*
     +                               (2*(1-5*dx*dx*xk1*xk1/12)*y(i-1)
     +                               -(1+dx*dx*xk2*xk2/12)*y(i-2))
                             if (abs(y(i)).gt.ymax) ymax=abs(y(i))
                     endif
             enddo
             return
             end
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
```

```
            function wav(x)
            common/one/ a,b,c,d,e
                if (x.ge.c.and.x.le.d) then
                    wav=a
            else if (x.ge.d.and.x.le.e) then
                    wav=b
            endif
            return
            end
```

VI. Appendix 3: Complete C Code for Numerov Program

```
/*************************************************************************/
/* Numerov algorithm, for PHY 232, by Stefan Zollner and Anand Shastri   */
/*************************************************************************/

/* Without these include statements, the C compiler is pretty dumb and does
not know how to calculate a square root or write to a file. */
#include <stdio.h>
#include <math.h>

/* define and initialize global variables (accessible in all functions) */
double xk1=0.0, xk2=0.0, xinit=0.0, xmid=0.0, xfin=0.0;

/* define function prototypes for functions at the end of the file. */
double wav(double);
void Numerov(double, double, double, double, int, double*, double[], double[]);
/*************************************************************************/
/* The main() function is next. */
int main(int argc, char *argv[])
{
   double x[10000], y[10000];
   double T=0.0, rho1=0.0, rho2=0.0, y0=0.0, Dy0=0.0, yL=0.0;
   int N=0, i=0;
   double pi=3.1415926, ymax=0.0, delta=0.0, freq=0.0;
   char line[100];
   FILE *input_file=NULL, *output_file=NULL;

/* reading the input file */
   input_file=fopen("in.txt","r"); /* open file for reading */

   for (i=0; i<3; i++) fgets(line,sizeof(line),input_file);
/* read one line and store it in array line[] */
   fgets(line,sizeof(line),input_file);
   sscanf(line,"%lf %lf %lf",&T, &rho1, &rho2);
   for (i=0; i<4; i++) fgets(line,sizeof(line),input_file);
   fgets(line,sizeof(line),input_file);
   sscanf(line,"%lf %lf %lf", &y0, &Dy0, &yL);
   for (i=0; i<4; i++) fgets(line,sizeof(line),input_file);
   fgets(line,sizeof(line),input_file);
   sscanf(line,"%lf",&xk1);
   for (i=0; i<4; i++) fgets(line,sizeof(line),input_file);
   fgets(line,sizeof(line),input_file);
   sscanf(line,"%lf %lf %lf %d",&xinit,&xmid,&xfin,&N);
   fclose(input_file);
/* performing the calculations */
   xk2=xk1*sqrt(rho2/rho1);
   Numerov(xinit,y0,Dy0,xfin,N,&ymax,x,y);
   delta=(y[N]-yL)/ymax;
   freq=sqrt(T/rho1)*xk1/2/pi;

   printf("Delta     = %lf\n",delta);
   printf("Frequency = %lf\n",freq);
   output_file=fopen("outs.txt","w");
```

```
   for (i=0; i<=N; i++) { fprintf(output_file,"%lf %lf\n",x[i],y[i]); }
   fclose(output_file);
   return 0;
};
/*************************************************************************/
/* The Numerov() function is next. */
void Numerov(double xi, double yi, double Dyi, double xf, int N,
             double *ymax, double x[], double y[])
{
   double dx=0.0, xk=0.0, xxk0=0.0, xxk1=0.0, xxk2=0.0; int i=0;
   dx=(xf-xi)/N;

   for (i=0; i<=N; i++) {
      if (i==0) {
         x[0]=xi; y[0]=yi;
         if (fabs(y[0])>*ymax) *ymax=fabs(y[0]);
      } else if (i==1) {
         x[1]=x[0]+dx; xk=wav(x[0]);
         y[1]=yi*(1-dx*dx*xk*xk/2)+Dyi*dx;
         if (fabs(y[1])>*ymax) *ymax=fabs(y[1]);
      } else {
         x[i]=x[i-1]+dx;
         xxk0=wav(x[i]); xxk1=wav(x[i]-dx); xxk2=wav(x[i]-2*dx);
         y[i]=1/(1+dx*dx*xxk0*xxk0/12)*(2*(1-5*dx*dx*xxk1*xxk1/12)*y[i-1]
                -(1+dx*dx*xxk2*xxk2/12)*y[i-2]);
         if (fabs(y[i])>*ymax) *ymax=fabs(y[i]);
      }
   }
   return;
}
/*************************************************************************/
/* Finally, the function wave(). */
/* return first mass density in first section of string,
   second mass density in second section of string, 0 outside */
double wav(double x)
{
   if ((x>=xinit) && (x<=xmid)) { return xk1; }
   else if ((x>=xmid) && (x<=xfin)) { return xk2; }
   else { return 0.0; }/* endif */
}
/*************************************************************************/
```